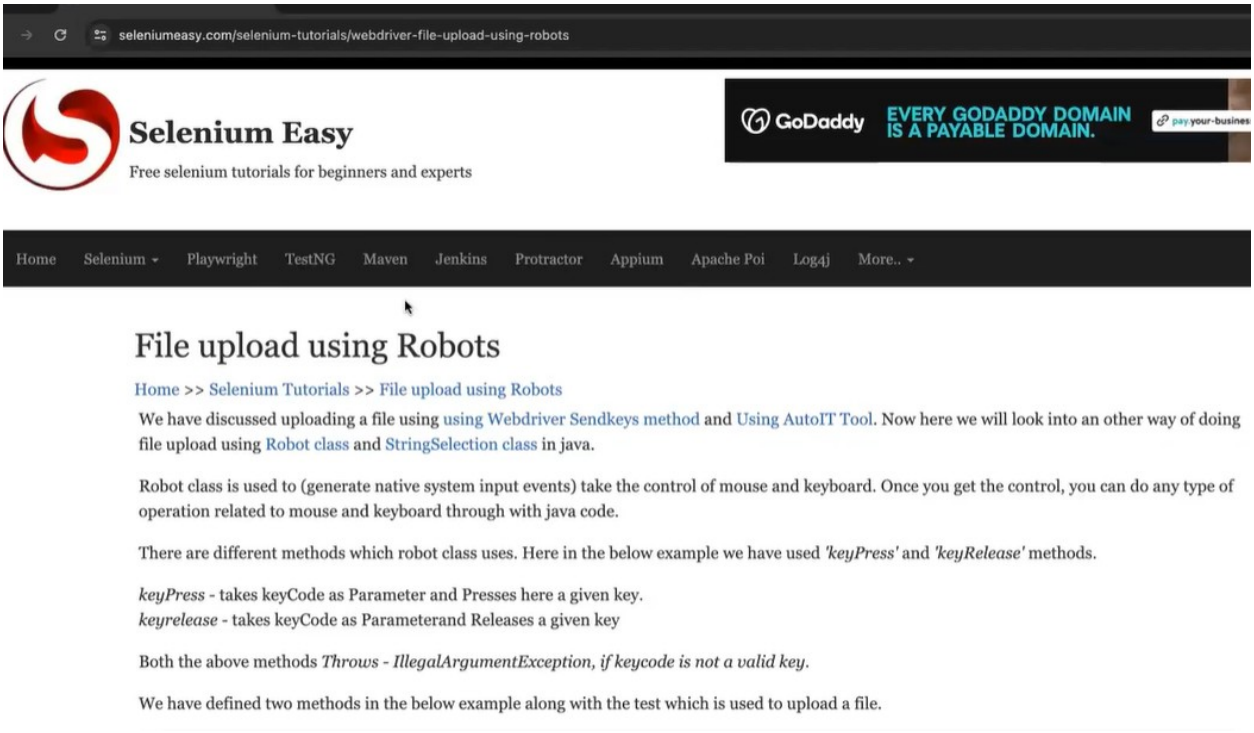
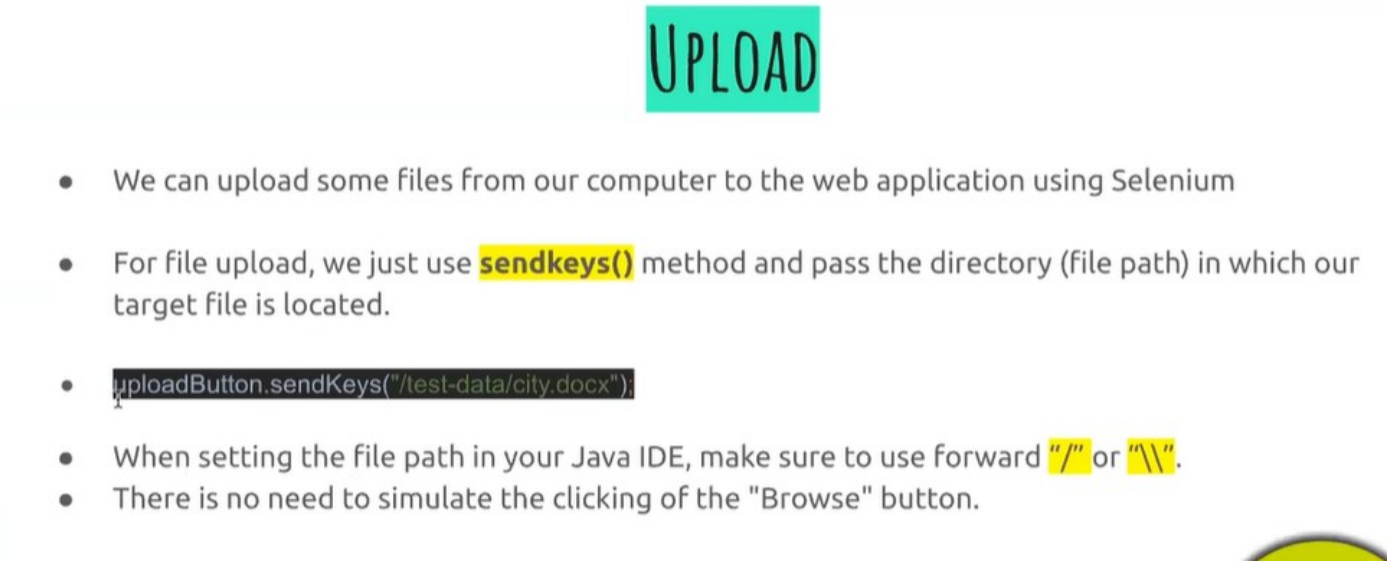
```
D. Uploading Files and Downloading Files
- The web applications sometimes might have the functionality to upload a file to it
- Some cases(common) will have that functionality done with html as input tag - we will handle with Selenium

- Some other cases there will be more complex upload strategies - you need to open OS native window, you drag and drop the file from
  - - Robot AWT class that comes as a Java API  can handle(automate) keystrokes and mouse actions on OS native level
  - - It will not be covered ion our sessions because it is outside of the scope

  1. Uploading a file with selenium:
  - Locate the input tag that will receive the file
  - use the element located and sendKeys() with file path to upload the file
  -- make sure your path to the file is dynamic with System.getProperty("user.dir")
  -- make sure the path uses appropriate file separators based on your OS - use Paths.get()

  2. Downloading a file
  - To change the download directory we do it at setup level when the driver is initialized
    In our case it is ion BrowserFactory
    -- create a map
    -- put the key as "download.default_directory" and value as path of your desired directory
    --- to get dynamically the path use above-mentioned strategies
    -- add this map to ChromeOptions object as options.setExperimentalOptions("prefs", map)
    -- add this ChromeOptions object to the driver instance constructor
       new ChromeDriver(options);
  - Next simply click on downloadable link or file or button.
  - After, we have different way to verify the file
```

File Upload using Robot (Will not be covered)


seleniumeasy.com/selenium-tutorials/webdriver-file-upload-using-robots
Selenium Easy
Free selenium tutorials for beginners and experts
GoDaddy
EVERY GODADDY DOMAIN IS A PAYABLE DOMAIN.
Home
Selenium
Playwright
TestNG
Maven
Jenkins
Protractor
Appium
Apache Poi
Log4j
More..
File upload using Robots
Home >> Selenium Tutorials >> File upload using Robots
We have discussed uploading a file using using Webdriver Sendkeys method and Using AutoIT Tool. Now here we will look into an other way of doing file upload using Robot class and StringSelection class in java.
Robot class is used to (generate native system input events) take the control of mouse and keyboard. Once you get the control, you can do any type of operation related to mouse and keyboard through with java code.
There are different methods which robot class uses. Here in the below example we have used 'keyPress' and 'keyRelease' methods.
keyPress - takes keyCode as Parameter and Presses here a given key.
keyrelease - takes keyCode as Parameterand Releases a given key
Both the above methods Throws - IllegalArgumentException, if keycode is not a valid key.
We have defined two methods in the below example along with the test which is used to upload a file.

```
    public static void uploadFile(String fileLocation) {
    try {
            //Setting clipboard with file location
        setClipboardData(fileLocation);
        //native key strokes for CTRL, V and ENTER keys
        Robot robot = new Robot();

        robot.keyPress(KeyEvent.VK_CONTROL);
        robot.keyPress(KeyEvent.VK_V);
        robot.keyRelease(KeyEvent.VK_V);
        robot.keyRelease(KeyEvent.VK_CONTROL);
        robot.keyPress(KeyEvent.VK_ENTER);
        robot.keyRelease(KeyEvent.VK_ENTER);
    } catch (Exception exp) {
            exp.printStackTrace();
    }
  }
}
```


UPLOAD
We can upload some files from our computer to the web application using Selenium
For file upload, we just use sendkeys() method and pass the directory (file path) in which our target file is located.
uploadButton.sendKeys("/test-data/city.docx");
When setting the file path in your Java IDE, make sure to use forward "/" or "\\".
There is no need to simulate the clicking of the "Browse" button.

Choose File inspection

Open the link : https://the-internet.herokuapp.com/

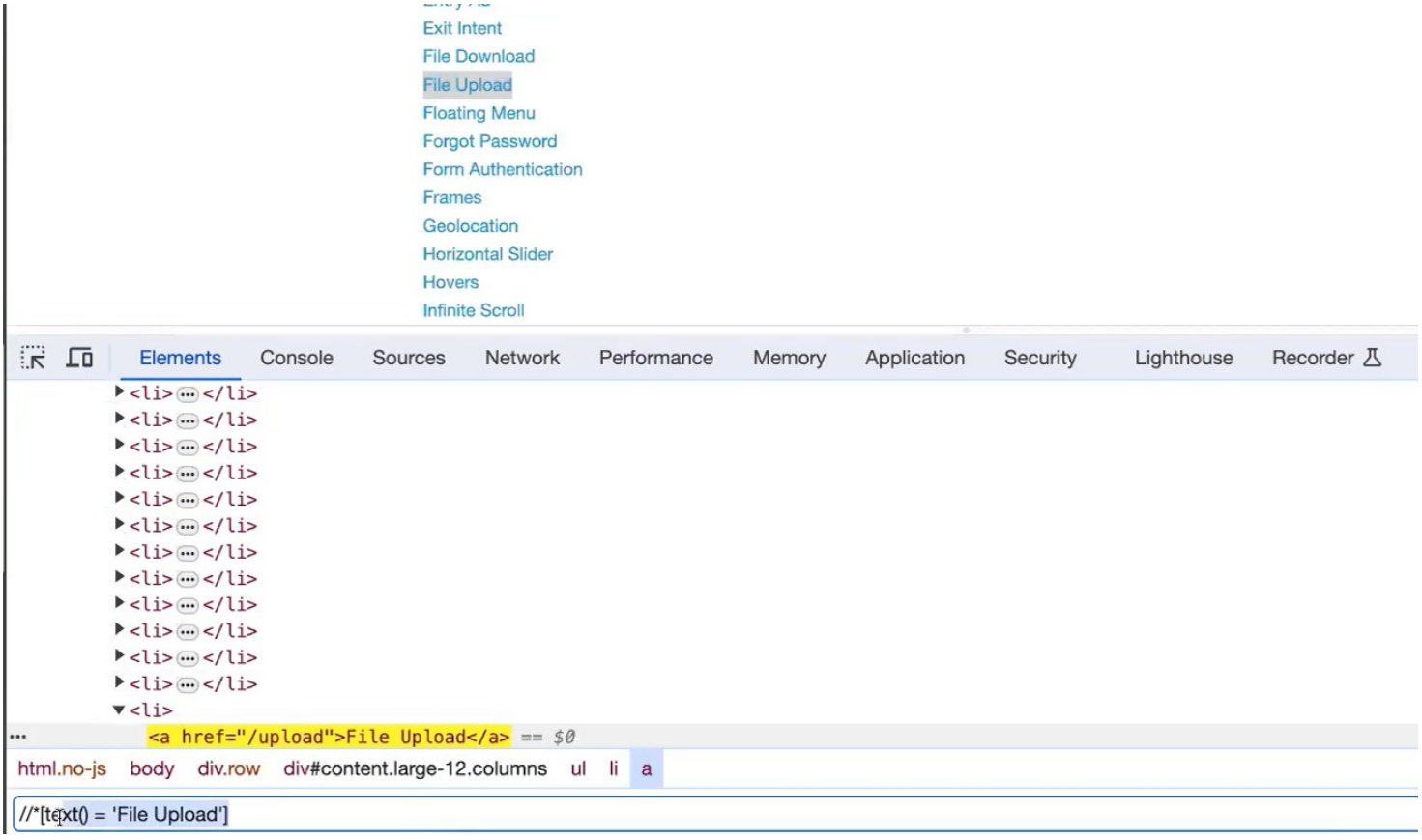
Click on: File Upload

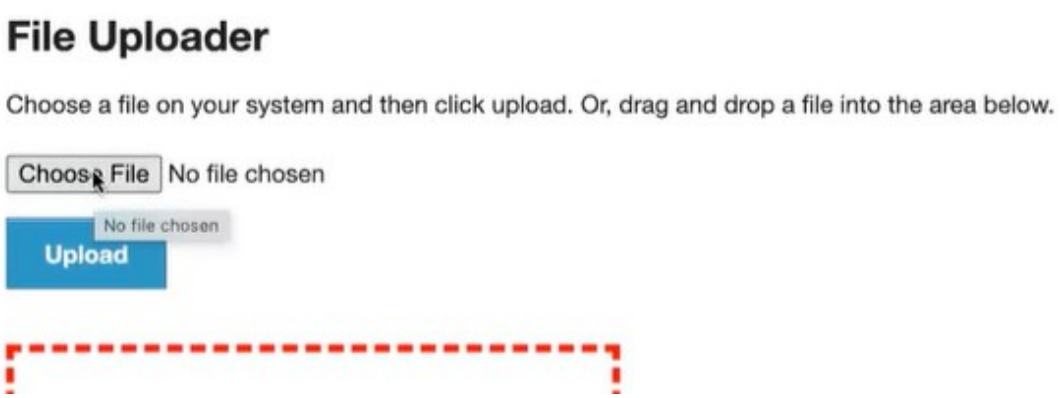
https://the-internet.herokuapp.com

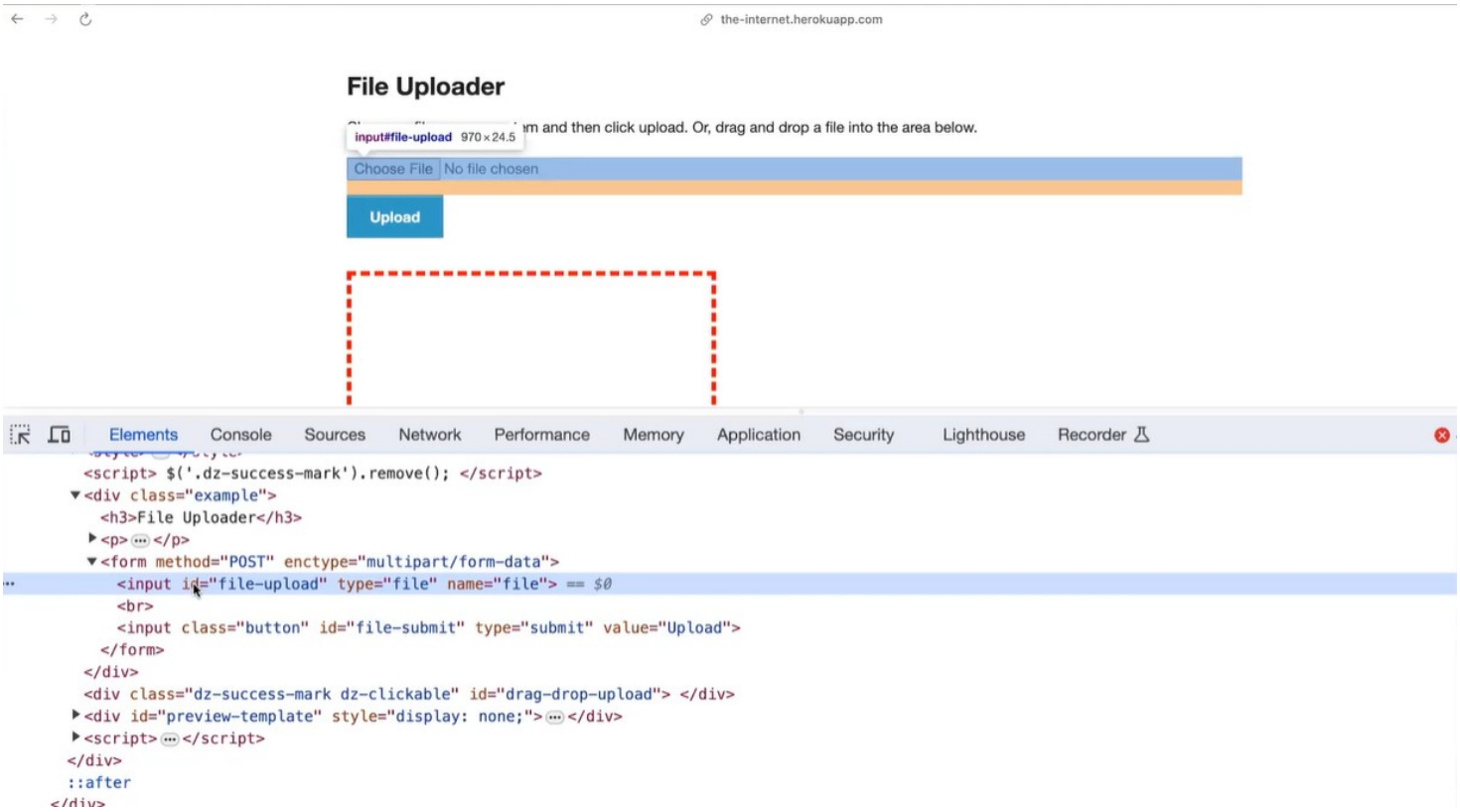
Welcome to the-internet

Available Examples

A/B Testing
Add/Remove Elements
Basic Auth (user and pass: admin)
Broken Images
Challenging DOM
Checkboxes
Context Menu
Digest Authentication (user and pass: admin)
Disappearing Elements
Drag and Drop
Dropdown
Dynamic Content
Dynamic Controls
Dynamic Loading
Entry Ad
Exit Intent
File Download
File Upload
Floating Menu


Exit Intent
File Download
File Upload
Floating Menu
Forgot Password
Form Authentication
Frames
Geolocation
Horizontal Slider
Hovers
Infinite Scroll
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
...
File Upload == $0
html.no-js body div.row div#content.large-12.columns ul li a
//*[text() = 'File Upload']


File Uploader
Choose a file on your system and then click upload. Or, drag and drop a file into the area below.
Choose File
No file chosen
No file chosen
Upload



Ways of calling ui-config.properties","herokuapp.url

```
  //First way
      //String url ="https://the-internet.herokuapp.com/";
      //Second Way USING ui-config.properties FILE:
      String url = ConfigurationReader.getProperty("ui-config.properties","herokuapp.url");
      //Third Way USING CONSTANTS VALUES IN INTERFACE FILE
```

```
package constants;

public interface Constants {
    String UI_CONFIG="ui-config.properties";
    String HEROKU_APP_URL="herokuapp.url";

      String USER_DIRECTORY="user.dir";


}
```

```
Feature: Uploading and Downloading Files With Selenium

  @upload
  Scenario: Upload a file to heroku app
    Given user navigates to heroku app file upload page
    When user uploads a file called "test image.png"
    Then the file is successfully uploaded
```

##File Uploadload URL
herokuapp.url:https://the-internet.herokuapp.com/

```java
package step_definitions;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.junit.Assert;
import pages.HerokuAppPage;
import utilities.Driver;
import utilities.WaitHelper;

public class HerokuAppPageSteps {
    private HerokuAppPage herokuAppPage= new HerokuAppPage();
    @Given("user navigates to heroku app file upload page")
    public void user_navigates_to_heroku_app_file_upload_page() {
        //First way
      //String url ="https://the-internet.herokuapp.com/";
      //Second Way USING ui-config.properties FILE:
      //String url = ConfigurationReader.getProperty("ui-
config.properties","herokuapp.url");
      //Third Way USING CONSTANTS VALUES IN INTERFACE FILE

String url =
ConfigurationReader.getProperty(Constants.UI_CONFIG,Constants.HEROKU_APP_URL);



        Driver.getInstance().getDriver().get(url);
        herokuAppPage.fileUploadLink.click();
        WaitHelper.shortWait();
    }
    @When("user uploads a file called {string}")
    public void user_uploads_a_file_called(String fileName) {

// To upload a file to an input tag we will use sendKeys() and we will provide the path of
the file
// Path of the file
// When you work with files and want to upload them you will need the whole path of it
```

```java
// 1. This path is not reliable - it will work only on my PC, if pushing this code to GitHub you will get an error when running the same scenario

String filePath = "/Users/dmytrochobotar/Developer/IntelliJ IDEA Projects/YFrameworkFinalC13/src/test/resources/test_data/person.png";
```

```java
//Uploading the file: Absolute Path:
The Path will work only on my Computer, I cannot push on GitHub
        String filePath="C:\\Users\\hicha\\IdeaProjects\\NewHrm\\src\\test\\
resources\\testData\\person.png";
We use instead : use.dir
```

```java
String basePath = System.getProperty("user.dir");
System.out.println("Base PATH:: " + basePath);
```

```java
//First way of using :use.dir
//String basePath= System.getProperty("use.dir");
//Second Way by using from CONSTANTS VALUES IN INTERFACE FILE:

String basePath= System.getProperty(Constants.USER_DIRECTORY);
```

Sytem.out.println(“Base PATH:::”+ basePath);

Final Corrected Way:

String filePath=basePath+”\\src\\test\\resources\\testData\\person.png”;

```
        herokuAppPage.choseFileButton.sendKeys(filePath);
        WaitHelper.shortWait();
        //Submit the file:
        herokuAppPage.uploadButton.click();
        WaitHelper.shortWait();
// \n, \t, \p, \", \\  sequence chars
// backslash in java is used to mark the following char, chars as special
// We can make java identify the file separator dynamically
// Option 1

// String filePath = basePathFile.separator + "src" +File.separator+ "test"
+File.separator+ "resources" +File.separator+ "test_data" +File.separator+ "person.png";
// Option 2
//String filePath
        Paths.get(basePath,  "src", "test", "resources", "test_data",
"person.png").toString();
        System.out.println("FULL PATH:::" + filePath);
// We need pass this path to sendKeys() methods
// Note we do not need to click on the button or do any other actions
// We simply send the file path to that input tag


    }
    @Then("the file is successfully uploaded")
    public void the_file_is_successfully_uploaded() {
        String expectedFileName ="person.png";
        String actualFileName= herokuAppPage.uploadedFilesResult.getText();
        Assert.assertEquals("File name verification
Failed",expectedFileName,actualFileName);
    }
}
```

```
package pages;

import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import utilities.Driver;

public class HerokuAppPage {
    public HerokuAppPage() {
        PageFactory.initElements(Driver.getInstance().getDriver(), this);
    }

    @FindBy(xpath = "//*[text()='File Upload']")
    public WebElement fileUploadLink;

    @FindBy(id = "file-upload")
    public WebElement choseFileButton;

    @FindBy(id = "file-submit")
    public WebElement uploadButton;
    @FindBy(id = "uploaded-files")
    public WebElement uploadedFilesResult;
```

```
}
```

```
package runner;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        plugin = {"pretty", "html:target/reports/cucumber-
reports.html","json:target/cucumber.json"},
        features = "src/test/resources/features",
        glue = "step_definitions",
        tags = "@upload",
        dryRun = false,
        monochrome = true
)
public class TestRunner {
    // No code is needed here; Cucumber will handle the execution
}
```

Another method by using fileName for uploading the png file:

```
    Feature: Uploading and Downloading Files With Selenium

  @upload
  Scenario: Upload a file to heroku app
    Given user navigates to heroku app file upload page
    When user uploads a file called "person.png"
    Then the file is successfully uploaded
```

```
package step_definitions;

import constants.Constants;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.junit.Assert;
import pages.HerokuAppPage;
import utilities.ConfigurationReader;
import utilities.Driver;
import utilities.WaitHelper;

import java.nio.file.Paths;

public class HerokuAppPageSteps {
     private String fileName;
    private HerokuAppPage herokuAppPage= new HerokuAppPage();
    @Given("user navigates to heroku app file upload page")
    public void user_navigates_to_heroku_app_file_upload_page() {
        //String url ="https://the-internet.herokuapp.com/";
```

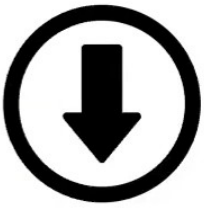
```
       // String url = ConfigurationReader.getProperty("ui-
config.properties","herokuapp.url");
        String url =
ConfigurationReader.getProperty(Constants.UI_CONFIG,Constants.HEROKU_APP_URL);

        Driver.getInstance().getDriver().get(url);
        herokuAppPage.fileUploadLink.click();
        WaitHelper.shortWait();
    }
    @When("user uploads a file called {string}")
    public void user_uploads_a_file_called(String fileName) {
     this.fileName=fileName;
// To upload a file to an input tag we will use sendKeys() and we will provide the path of
the file
// Path of the file
// When you work with files and want to upload them you will need the whole path of it
        //Uploading the file: Absolute Path:
       // String filePath="C:\\Users\\hicha\\IdeaProjects\\NewHrm\\src\\test\\resources\\
testData\\person.png";
        //herokuAppPage.choseFileButton.sendKeys(filePath);
        //String basePath= System.getProperty("user.dir");
        String basePath= System.getProperty(Constants.USER_DIRECTORY);
        String filePath=basePath+"\\src\\test\\resources\\testData\\person.png";
        herokuAppPage.choseFileButton.sendKeys(filePath);

        System.out.println("Base PATH:::"+ basePath);
        //Final Corrected Way:
        //String path=basePath+"\\src\\test\\resources\\testData\\person.png”;

        WaitHelper.shortWait();
        //Submit the file:
        herokuAppPage.uploadButton.click();
        WaitHelper.shortWait();

// \n, \t, \p, \", \\  sequence chars
// backslash in java is used to mark the following char, chars as special
// We can make java identify the file separator dynamically
// Option 1

// String filePath = basePathFile.separator + "src" +File.separator+ "test"
+File.separator+ "resources" +File.separator+ "test_data" +File.separator+ "person.png";
        // Option 2
        //String filePath
        //Paths.get(basePath,  "src", "test", "resources", "test_data",
"person.png").toString();
        // Option 3
        //String filePath
        Paths.get(basePath,  "src", "test", "resources", "test_data",
fileName).toString();
        System.out.println("FULL PATH:::" + filePath);
// We need pass this path to sendKeys() methods
// Note we do not need to click on the button or do any other actions
// We simply send the file path to that input tag
        //herokuAppPage.choseFileButton.sendKeys(filePath);
        WaitHelper.shortWait();

    }
    @Then("the file is successfully uploaded")
    public void the_file_is_successfully_uploaded() {
        //String expectedFileName ="person.png";
         String expectedFileName =fileName;
```

```
        String actualFileName= herokuAppPage.uploadedFilesResult.getText();
        Assert.assertEquals("File name verification
Failed",expectedFileName,actualFileName);
    }
}
```

DOWNLOAD

- Selenium can't download file by default, since clicking on download prompts browser window to show up.
  - Selenium cannot interact with browser download popup.
  - Therefore extra configuration needed on browser level, to bypass that step.

- We use browser options (Chrome or Firefox) to set default file download location and disable download popup.
- For chrome disabling download popup is no longer needed.



```
WebDriverManager.chromedriver().setup();
HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
chromePrefs.put("profile.default_content_settings.popups", 0);
chromePrefs.put("download.default_directory",
System.getProperty("user.dir")+"\\src\\test\\resources\\testData\\Downloads");
ChromeOptions options = new ChromeOptions();
options.setExperimentalOption("prefs", chromePrefs);
driver = new ChromeDriver(options);
```

Click on file Download:

nternet.herokuapp.com

Available Examples

A/B Testing
Add/Remove Elements
Basic Auth (user and pass: admin)
Broken Images
Challenging DOM
Checkboxes
Context Menu
Digest Authentication (user and pass: admin)
Disappearing Elements
Drag and Drop
Dropdown
Dynamic Content
Dynamic Controls
Dynamic Loading
Entry Ad
Exit Intent
File Download

https://the-internet.herokuapp.com/download

File Downloader

myfile.pdf
Кота арестовали.jpeg
Homework2.png
selenium-snapshot.png
b.jpg
some-file.txt
FileUpload.txt
UploadFileExample.txt
random_data.txt
C VPN Ñekoray-3.png
person.png
pom.xml
Test1.pdf
loginCred.jpg
TestData.txt
createlead.png
LambdaTest.txt
file_for_upload.jpg
69st002u9q_Screenshot (2).png
Про будущее нейросетей.jpeg
Test2.pdf
chatgptprompt1.png

```
Feature: Uploading and Downloading Files With Selenium

  Background: Set Up
    Given user navigates to heroku app
  @upload
  Scenario: Upload a file to heroku app
    And clicks on file upload page
    When user uploads a file called "person.png"
    Then the file is successfully uploaded

  @downloadNHrmProject
  Scenario: Download a file from heroku app
    And clicks on file download page
    When user clicks on "text2.txt"
    Then the file is successfully downloaded
```

```
package runners;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        plugin = {"pretty","html:target/cucumber", "json:target/cucumber.json"},
        features="src/test/resources/uiFeatures",
        glue="ui_automation.step_definitions",
        tags="@downloadNHrmProject",
        dryRun = false,
        monochrome = true
)

public class UITestRunner {
```

```
}
```

```
package ui_automation.pages;

import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import ui_automation.utilities.Driver;

public class HerokuAppPage {
    public HerokuAppPage() {
        PageFactory.initElements(Driver.getInstance().getDriver(), this);
    }

    @FindBy(xpath = "//*[text()='File Upload']")
    public WebElement fileUploadLink;

    @FindBy(id = "file-upload")
    public WebElement choseFileButton;

    @FindBy(id = "file-submit")
    public WebElement uploadButton;

    @FindBy(xpath = "//*[text()='File Download']")
    public WebElement uploadedFilesResult;

    @FindBy(xpath = "//*[text()='File Download']")
    public WebElement fileDownloadLink;

//    @FindBy(xpath = "//*[text()='LambdaTest.txt']")
//    public WebElement LambdaTestTextFileLink;

    @FindBy(xpath = "//*[text()='text2.txt']")
    public WebElement Text2FileLink;

}
```

```
package ui_automation.step_definitions;

import constants.Constants;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.apache.commons.io.FileUtils;
import org.junit.Assert;
import org.openqa.selenium.WebDriver;
import ui_automation.pages.HerokuAppPage;
import ui_automation.utilities.ConfigurationReader;
import ui_automation.utilities.Driver;
import ui_automation.utilities.FileHelper;
import ui_automation.utilities.WaitHelper;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
```

```
import java.nio.file.Paths;

public class HerokuAppPageSteps {
    private WebDriver driver = Driver.getInstance().getDriver();
    private String downloadFileName;
    private String uploadFileName;
    private HerokuAppPage herokuAppPage = new HerokuAppPage();

  //=========UPLOAD======
    @Given("user navigates to heroku app")
    public void user_navigates_to_heroku_app() {
        String url = ConfigurationReader.getProperty(Constants.UI_CONFIG,
Constants.HEROKU_APP_URL);
        Driver.getInstance().getDriver().get(url);
    }

    @Given("clicks on file upload page")
    public void user_clicks_on_file_upload_page() {
        herokuAppPage.fileUploadLink.click();
        WaitHelper.shortWait();
    }

    @When("user uploads a file called {string}")
    public void user_uploads_a_file_called(String fileName) {
        this.uploadFileName = fileName;

        String basePath = System.getProperty(Constants.USER_DIRECTORY);
        String filePath = Paths.get(basePath, "src", "test", "resources", "testData",
fileName).toString();
        herokuAppPage.choseFileButton.sendKeys(filePath);
        herokuAppPage.uploadButton.click();
        WaitHelper.shortWait();
    }

    @Then("the file is successfully uploaded")
    public void the_file_is_successfully_uploaded() {
        String expectedFileName= uploadFileName;
        String actualFileName = herokuAppPage.uploadedFilesResult.getText();
        Assert.assertEquals("File name verification failed", expectedFileName,
actualFileName);
    }
//=========DOWNLOAD======
    @Given("clicks on file download page")
    public void clicks_on_file_download_page() {
        herokuAppPage.fileDownloadLink.click();
        WaitHelper.shortWait();
    }

    @When("user clicks on {string}")
    public void user_clicks_on(String fileName) {
        this.downloadFileName = fileName;
        herokuAppPage.LambdaTestTextFileLink.click();
        WaitHelper.shortWait();
    }

    @Then("the file is successfully downloaded")
    public void the_file_is_successfully_downloaded() {
        String basePath = System.getProperty(Constants.USER_DIRECTORY);
        String filePath =  Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", downloadFileName).toString();
```

```
        File expectedFile = new File(filePath);
        boolean fileExists =expectedFile.exists() && expectedFile.isFile();
        Assert.assertTrue("The file existence verification failed.", fileExists);
    }



   @Then("the file content matches the {string}")
 public void the_file_content_matches_the(String expectedFileName) {

  }

}
```

```
package ui_automation.utilities;

import com.google.common.base.Function;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.openqa.selenium.*;
import org.openqa.selenium.support.ui.*;

import java.time.Duration;

public class WaitHelper {

    private static final Logger oLog = LogManager.getLogger(WaitHelper.class);


    public static void wait(int secs) {
        try {
            Thread.sleep(1000 * secs);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void waitUntilClickable (WebElement element, int
timeToWaitInSec) {
        Wait wait = new WebDriverWait(Driver.getInstance().getDriver(),
timeToWaitInSec);
        wait.until(ExpectedConditions.elementToBeClickable(element));
    }
    // Short Wait
     public static void shortWait() {
        System.out.println("Short wait has been used");
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    public static WebElement waitForVisibility(WebElement element, int
timeToWaitInSec) {
```

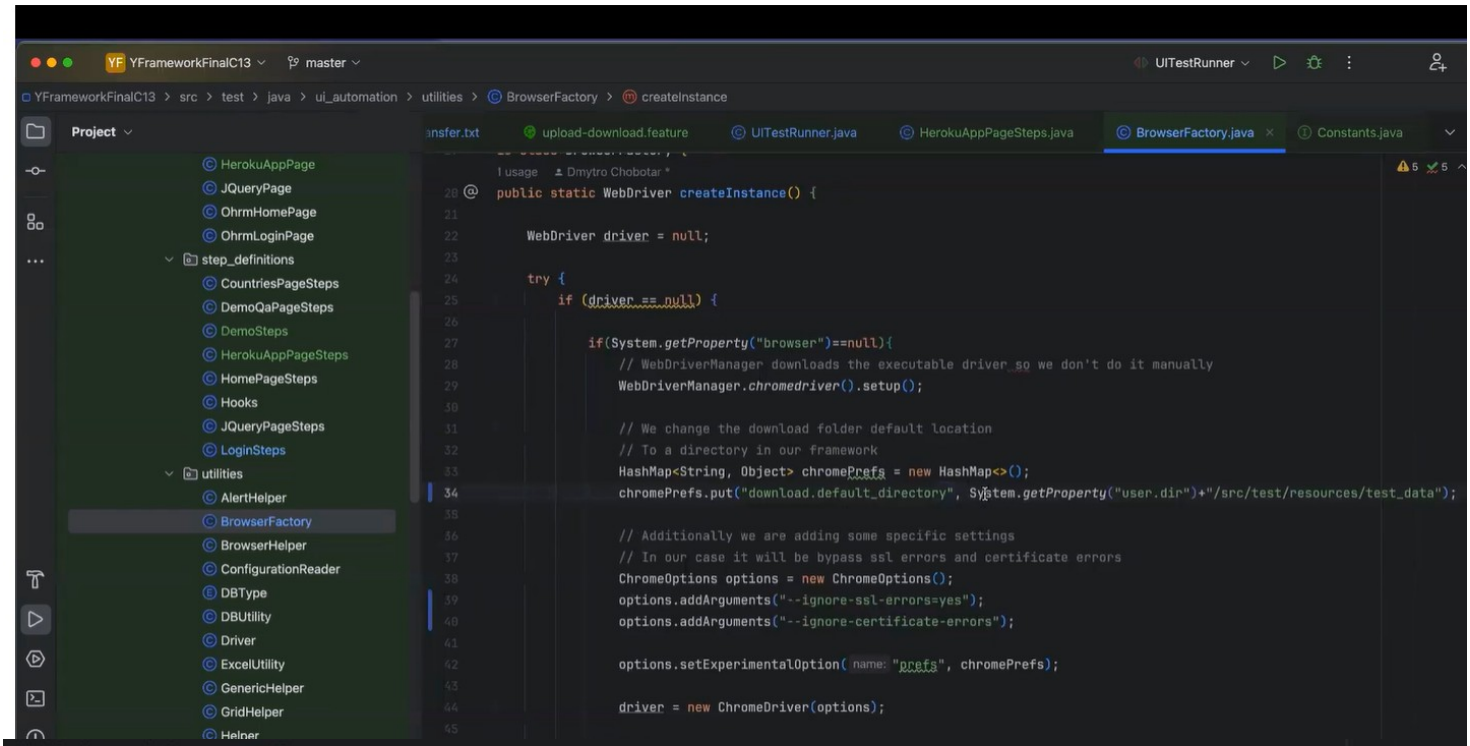
```
        WebDriverWait wait = new WebDriverWait(Driver.getInstance().getDriver(),
timeToWaitInSec);
        return wait.until(ExpectedConditions.visibilityOf(element));
    }

    public static WebElement waitForClickablility(WebElement element, int timeout)
{
        WebDriverWait wait = new WebDriverWait(Driver.getInstance().getDriver(),
timeout);
        return wait.until(ExpectedConditions.elementToBeClickable(element));
    }

    public static void waitForPageToLoad(long timeOutInSeconds) {
        ExpectedCondition<Boolean> expectation = new ExpectedCondition<Boolean>()
{
            public Boolean apply(WebDriver driver) {
                return ((JavascriptExecutor) driver).executeScript("return
document.readyState").equals("complete");
            }
        };
        try {
            System.out.println("Waiting for page to load...");
            WebDriverWait wait = new
WebDriverWait(Driver.getInstance().getDriver(), timeOutInSeconds);
            wait.until(expectation);
        } catch (Throwable error) {
            System.out.println(
                    "Timeout waiting for Page Load Request to complete after " +
timeOutInSeconds + " seconds");
        }
    }

    public static WebElement fluentWait(final WebElement webElement, int
timeinsec) {
        FluentWait<WebDriver> wait = new
FluentWait<WebDriver>(Driver.getInstance().getDriver())
                .withTimeout(Duration.ofSeconds(timeinsec))
                .pollingEvery(Duration.ofMillis(500))
                .ignoring(NoSuchElementException.class);
        WebElement element = wait.until(new Function<WebDriver, WebElement>() {
            public WebElement apply(WebDriver driver) {
                return webElement;
            }
        });
        return element;
    }

    public void hardWait(int timeOutInMiliSec) throws InterruptedException {
        oLog.info(timeOutInMiliSec);
        Thread.sleep(timeOutInMiliSec);
    }

    public WebElement handleStaleElement(By locator,int retryCount,int
delayInSeconds) throws InterruptedException {
        oLog.info(locator);
        WebElement element = null;
```

```
        while (retryCount >= 0) {
            try {
                element = Driver.getInstance().getDriver().findElement(locator);
                return element;
            } catch (StaleElementReferenceException e) {
                hardWait(delayInSeconds);
                retryCount--;
            }
        }
        throw new StaleElementReferenceException("Element cannot be recovered");
    }
}
```

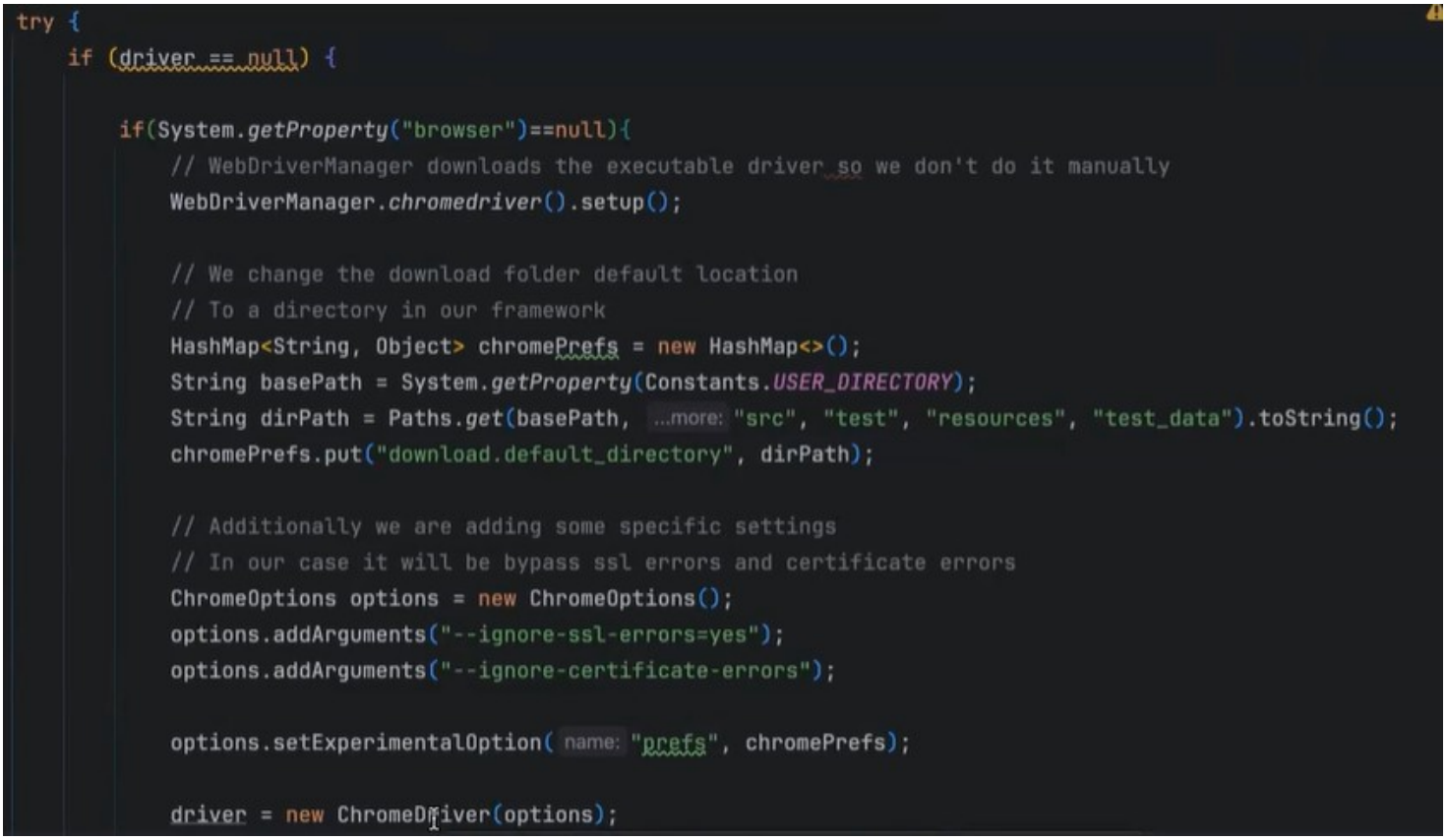
```
package ui_automation.utilities;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebDriverException;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.edge.EdgeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.openqa.selenium.safari.SafariDriver;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.HashMap;
public class BrowserFactory {
```

Another method for uploading

Selenium - Upload and Download, Logging - Jan 21st 2024




```
WebDriver driver = null;

try {
    if (driver == null) {

        if(System.getProperty("browser")==null){
            // WebDriverManager downloads the executable driver so we don't do it manually
            WebDriverManager.chromedriver().setup();

            // We change the download folder default location
            // To a directory in our framework
            HashMap<String, Object> chromePrefs = new HashMap<>();
            chromePrefs.put("download.default_directory", System.getProperty("user.dir")+"/src/test/resources/test_data");

            // Additionally we are adding some specific settings
            // In our case it will be bypass ssl errors and certificate errors
            ChromeOptions options = new ChromeOptions();
            options.addArguments("--ignore-ssl-errors=yes");
            options.addArguments("--ignore-certificate-errors");

            options.setExperimentalOption( name: "prefs", chromePrefs);

            driver = new ChromeDriver(options);
```



```
try {
    if (driver == null) {

        if(System.getProperty("browser")==null){
            // WebDriverManager downloads the executable driver so we don't do it manually
            WebDriverManager.chromedriver().setup();

            // We change the download folder default location
            // To a directory in our framework
            HashMap<String, Object> chromePrefs = new HashMap<>();
            String basePath = System.getProperty(Constants.USER_DIRECTORY);
            String dirPath = Paths.get(basePath, ...more: "src", "test", "resources", "test_data").toString();
            chromePrefs.put("download.default_directory", dirPath);

            // Additionally we are adding some specific settings
            // In our case it will be bypass ssl errors and certificate errors
            ChromeOptions options = new ChromeOptions();
            options.addArguments("--ignore-ssl-errors=yes");
            options.addArguments("--ignore-certificate-errors");

            options.setExperimentalOption( name: "prefs", chromePrefs);

            driver = new ChromeDriver(options);
```

```java
public class BrowserFactory {
    /**
     * the createInstance method will create different browser instances
     * based on the need.
     * If we execute through the command line like below, by specifying the type
of browser
     * `mvn clean test -Dtest=UITestRunner -Dbrowser=chromeRemote`
     * if we don't specify the type of browser from command line, or execute
though IDE,
     * JVM won't have browser property
     * <p>
     * mvn clean test -Dtest=UITestRunner
     *
     * @return WebDriver instance
     */
    public static WebDriver createInstance() {

        WebDriver driver = null;

        try {
            if (System.getProperty("browser") == null) {
                // because of the selenium version 3, we need to specify where is
the exec file
                // Boni Garcia Library is profiding below line to find out during
runtime
                WebDriverManager.chromedriver().setup();
                // Create a key/value of ther preferences
                HashMap<String, Object> chromePrefs = new HashMap<String,
Object>();
```

```
                String path = System.getProperty("user.dir") +
                        File.separator +
                        "src" +
                        File.separator +
                        "test" +
                        File.separator +
                        "resources" +
                        File.separator +
                        "testData" +
                        File.separator +
                        "Downloads";
                chromePrefs.put("download.default_directory", path);
                // create the Chrome Options
                ChromeOptions options = new ChromeOptions();
                options.addArguments("--ignore-ssl-errors=yes");
                options.addArguments("--ignore-certificate-errors");
                options.setExperimentalOption("prefs", chromePrefs);
                driver = new ChromeDriver(options);
            } else {
                switch (System.getProperty("browser")) {
                    case "chrome-headless":
                        WebDriverManager.chromedriver().setup();
                        driver = new ChromeDriver(new
ChromeOptions().setHeadless(true));
                        break;
                    case "chromeRemote":
                        HashMap<String, Object> chromePrefs = new HashMap<String,
Object>();
                          chromePrefs.put("download.default_directory",
System.getProperty("user.dir") + "\\src\\test\\resources\\testData\\
Downloads");
                        ChromeOptions chrOptions = new ChromeOptions();
                        chrOptions.addArguments("--ignore-ssl-errors=yes");
                        chrOptions.addArguments("--ignore-certificate-errors");
                        chrOptions.setExperimentalOption("prefs", chromePrefs);
                        try {
                            driver = new RemoteWebDriver(new
URL("http://54.80.58.108:4444/wd/hub"), chrOptions);
                        } catch (MalformedURLException e) {
                            e.printStackTrace();
                        }
                        break;
                    case "firefox":
                        WebDriverManager.firefoxdriver().setup();
                        driver = new FirefoxDriver();
                        break;
                    case "firefox-headless":
                        WebDriverManager.firefoxdriver().setup();
                        driver = new FirefoxDriver(new
FirefoxOptions().setHeadless(true));
                        break;
                    case "firefoxRemote":
                        FirefoxOptions firOptions = new FirefoxOptions();
                        try {
```

```
                            driver = new RemoteWebDriver(new
URL("http://54.80.58.108:4444/wd/hub"), firOptions);
                        } catch (MalformedURLException e) {
                            e.printStackTrace();
                        }
                        break;
                    case "ie":
                        if
(System.getProperty("os.name").toLowerCase().contains("mac")) {
                            throw new WebDriverException("Your operating system
does not support the requested browser");
                        }
                        WebDriverManager.iedriver().setup();
                        driver = new InternetExplorerDriver();
                        break;

                    case "edge":
                        if
(System.getProperty("os.name").toLowerCase().contains("mac")) {
                            throw new WebDriverException("Your operating system
does not support the requested browser");
                        }
                        WebDriverManager.edgedriver().setup();
                        driver = new EdgeDriver();
                        break;

                    case "safari":
                        if
(System.getProperty("os.name").toLowerCase().contains("windows")) {
                            throw new WebDriverException("Your operating system
does not support the requested browser");
                        }
                        WebDriverManager.getInstance(SafariDriver.class).setup();
                        driver = new SafariDriver();
                        break;
                    default:
                        WebDriverManager.chromedriver().setup();
                        driver = new ChromeDriver();
                        break;
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return driver;
    }

    public static WebDriver createInstance(String browser) {
        WebDriver driver = null;
        if (browser.equals("chromeRemote")) {
            HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
```

```
            chromePrefs.put("download.default_directory",
System.getProperty("user.dir") + "\\src\\test\\resources\\testData\\
Downloads");
            ChromeOptions chrOptions = new ChromeOptions();
            chrOptions.addArguments("--ignore-ssl-errors=yes");
            chrOptions.addArguments("--ignore-certificate-errors");
            chrOptions.setExperimentalOption("prefs", chromePrefs);

            try {
                driver = new RemoteWebDriver(new
URL("http://54.80.58.108:4444/wd/hub"), chrOptions);
            } catch (MalformedURLException e) {
                e.printStackTrace();
            }
        } else {
            driver = new ChromeDriver();
        }
        return driver;
    }
}
```

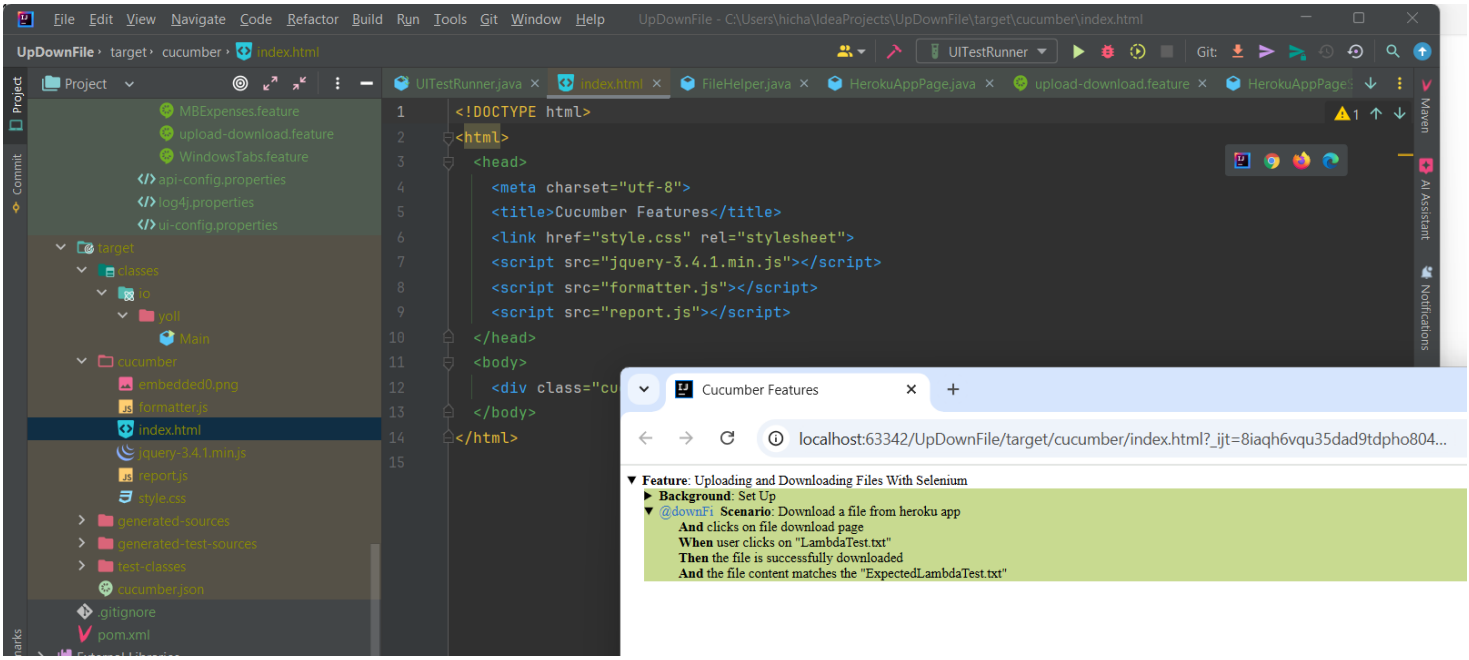
```
package ui_automation.utilities;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
public class ConfigurationReader {
    //Encapsulation logic
    private static Properties properties;

    public static String getProperty(String fileName, String key) {
        try {
            String path = "src/test/resources/"+ fileName;
            //FileInputStream is Java class and will read the file
            FileInputStream stream = new FileInputStream(path);
            properties = new Properties();
            //this will load the value from key
            properties.load(stream);
            stream.close();
        }catch (IOException e) {
            e.printStackTrace();
        }
        return properties.getProperty(key);
    }
}
```

```
package ui_automation.utilities;

import org.openqa.selenium.WebDriver;

public class Driver {
    /* private constructor, no objects can be created outside of this class */
    private Driver(){

    }

    /* private static reference variable to the class */
```

```
    private static Driver instance=new Driver();

    /* create a getter method with return type of the class => encapsulation */
    public static Driver getInstance(){
        return instance;
    }

    /**
     * Thread Local is responsible to keep track of the webdrivers that has been created by
different threads
     * thread local driver object for webdriver
     */
    ThreadLocal<WebDriver> driver=new ThreadLocal<WebDriver>();

    /* getDriver() is the instance method of our Driver class */
    public WebDriver getDriver(){
        return driver.get();
    }

    /* setDriver() is the instance method of our Driver class */
    public  void setDriver(WebDriver driverParameter){

        driver.set(driverParameter);
    }

    /* removeDriver() is the instance method of our Driver class */
    public void removeDriver(){
        driver.get().quit();
        driver.remove();
    }
}
```

```
package constants;

public interface Constants {
           String UI_CONFIG="ui-config.properties";
           String HEROKU_APP_URL="herokuapp.url";
           String USER_DIRECTORY="user.dir";
}
```

Report :

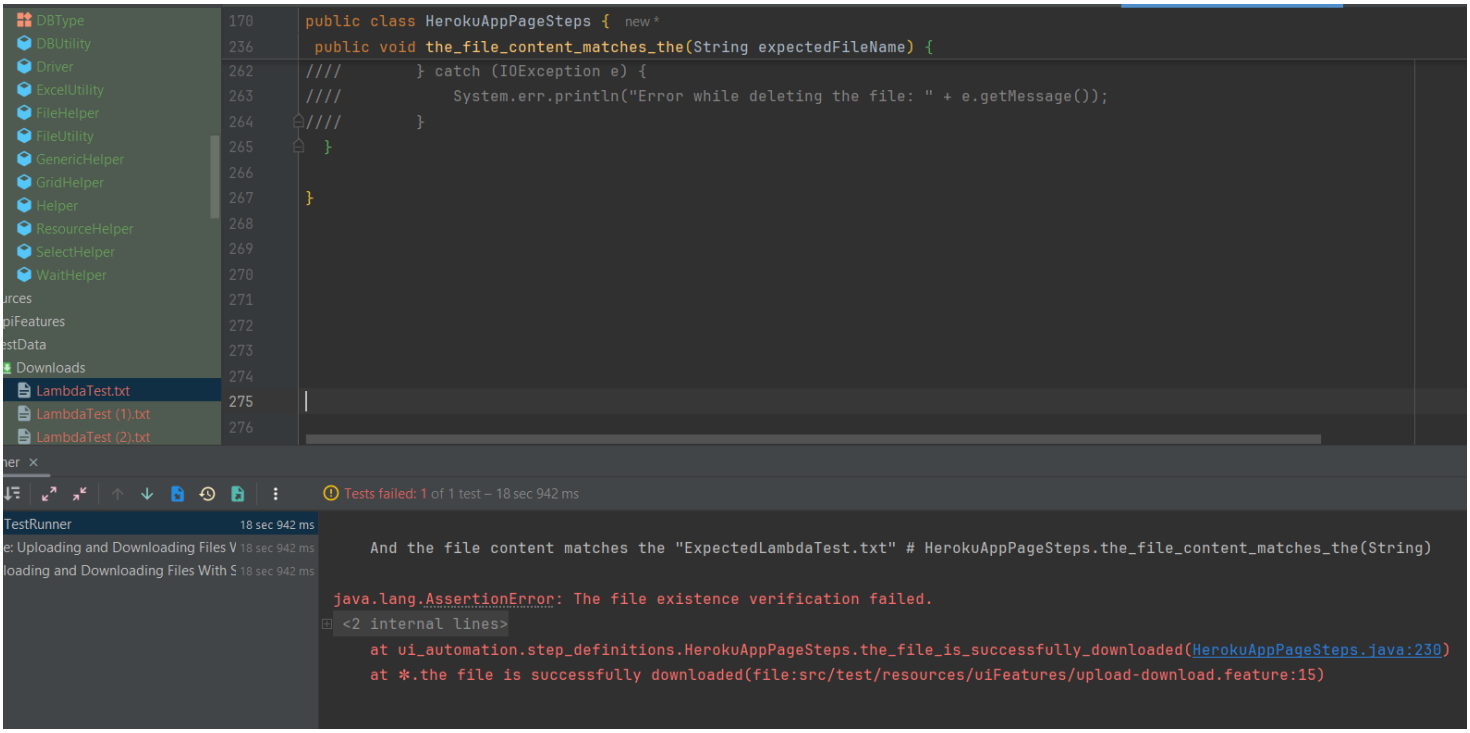


How to see there is an error after assertion When downloading the file:

Example

LambdaTest1.txt ## LambdaTest.txt

```
@downFi
Scenario: Download a file from heroku app
  And clicks on file download page
  When user clicks on "LambdaTest1.txt"
  Then the file is successfully downloaded
```



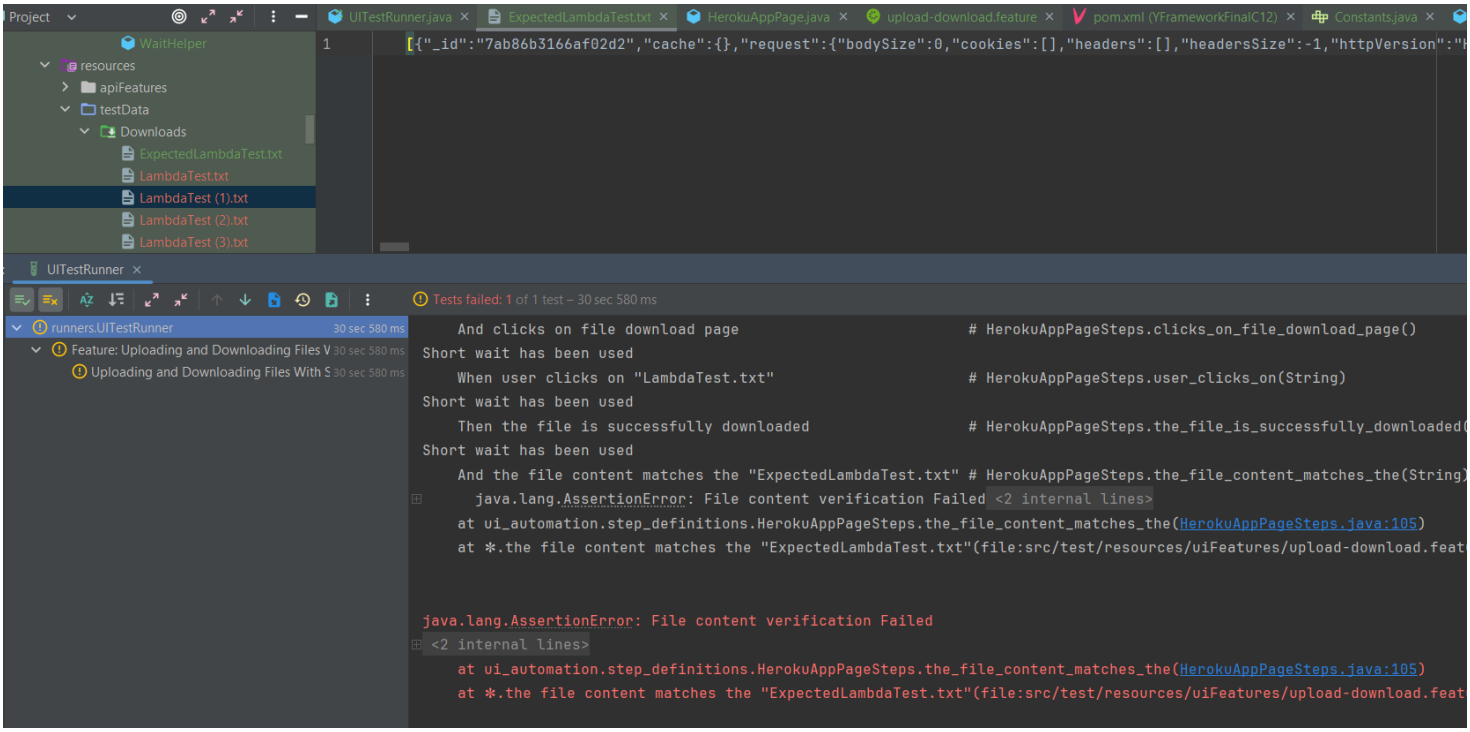
How to see there is an error after assertion When downloading the file:

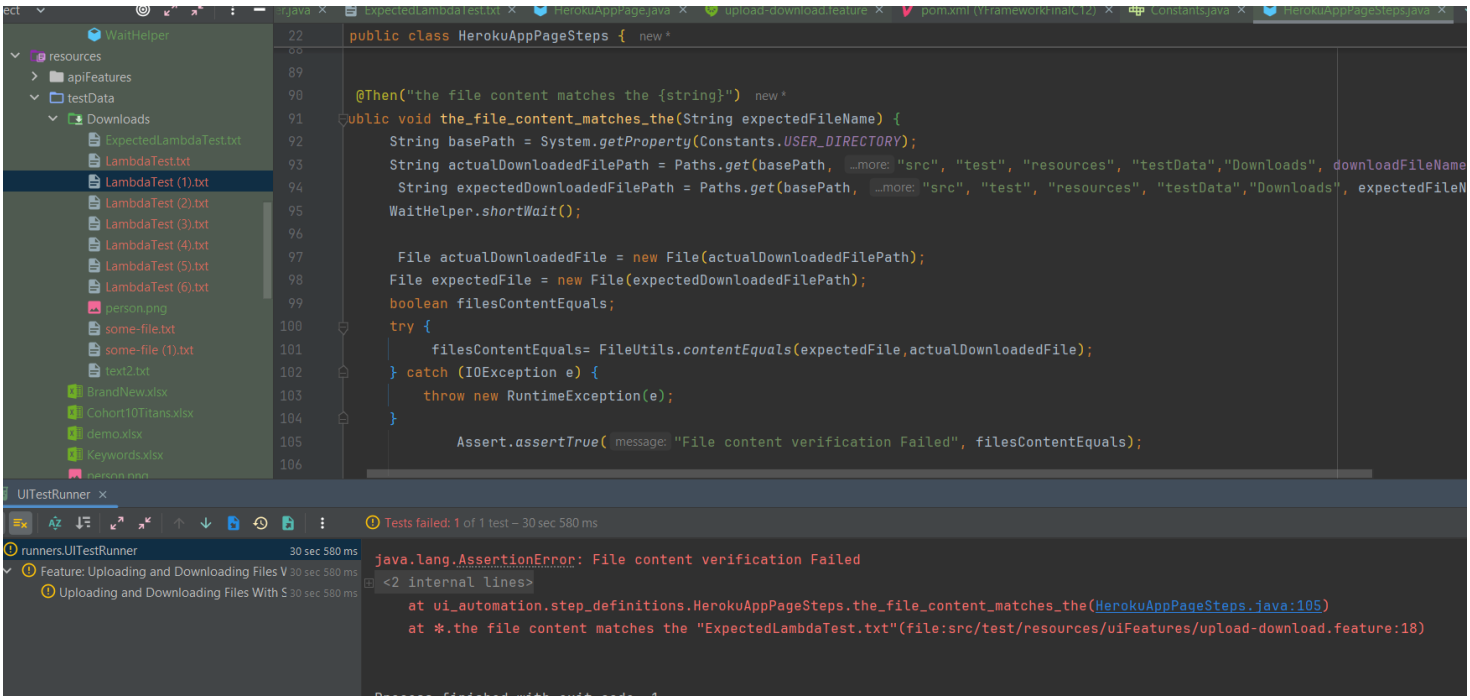
First: you have to have ExpectedLambdaTest.txt on the Project

Example

ExpectedLambdaTest.txt ## ExpectedLambdaTest.txt

```
@downFi
Scenario: Download a file from heroku app
  And clicks on file download page
  When user clicks on "LambdaTest.txt"
  #When user clicks on "some-file.txt"

  Then the file is successfully downloaded
  And the file content matches the "ExpectedLambdaTest.txt"
```




public class HerokuAppPageSteps { new *
@Then("the file content matches the {string}") new *
public void the_file_content_matches_the(String expectedFileName) {
String basePath = System.getProperty(Constants.USER_DIRECTORY);
String actualDownloadedFilePath = Paths.get(basePath, ...more: "src", "test", "resources", "testData","Downloads", downloadFileName
String expectedDownloadedFilePath = Paths.get(basePath, ...more: "src", "test", "resources", "testData","Downloads", expectedFileN
WaitHelper.shortWait();
File actualDownloadedFile = new File(actualDownloadedFilePath);
File expectedFile = new File(expectedDownloadedFilePath);
boolean filesContentEquals;
try {
filesContentEquals= FileUtils.contentEquals(expectedFile,actualDownloadedFile);
} catch (IOException e) {
throw new RuntimeException(e);
}
Assert.assertTrue(message: "File content verification Failed", filesContentEquals);
UITestRunner
Tests failed: 1 of 1 test – 30 sec 580 ms
runners.UITestRunner 30 sec 580 ms
Feature: Uploading and Downloading Files V 30 sec 580 ms
Uploading and Downloading Files With S 30 sec 580 ms
java.lang.AssertionError: File content verification Failed
<2 internal lines>
at ui_automation.step_definitions.HerokuAppPageSteps.the_file_content_matches_the(HerokuAppPageSteps.java:105)
at *.the file content matches the "ExpectedLambdaTest.txt"(file:src/test/resources/uiFeatures/upload-download.feature:18)

```
Feature: Uploading and Downloading Files With Selenium

  Background: Set Up
    Given user navigates to heroku app
  @upload
  Scenario: Upload a file to heroku app
    And clicks on file upload page
    When user uploads a file called "person.png"
    Then the file is successfully uploaded

  @downFi
  Scenario: Download a file from heroku app
```

```
    And clicks on file download page
    When user clicks on "LambdaTest.txt"
    #When user clicks on "some-file.txt"

    Then the file is successfully downloaded
    And the file content matches the "ExpectedLambdaTest.txt"
    #And the file content matches the "ExpectedSome-file.txt"
```

```
package ui_automation.step_definitions;

import constants.Constants;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.apache.commons.io.FileUtils;
import org.junit.Assert;
import org.openqa.selenium.WebDriver;
import ui_automation.pages.HerokuAppPage;
import ui_automation.utilities.ConfigurationReader;
import ui_automation.utilities.Driver;
import ui_automation.utilities.FileHelper;
import ui_automation.utilities.WaitHelper;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class HerokuAppPageSteps {
    private WebDriver driver = Driver.getInstance().getDriver();
    private String downloadFileName;
    private String uploadFileName;
    private HerokuAppPage herokuAppPage = new HerokuAppPage();

    @Given("user navigates to heroku app")
    public void user_navigates_to_heroku_app() {
        String url = ConfigurationReader.getProperty(Constants.UI_CONFIG,
Constants.HEROKU_APP_URL);
        Driver.getInstance().getDriver().get(url);
        WaitHelper.shortWait();

    }

    @Given("clicks on file upload page")
    public void user_clicks_on_file_upload_page() {
        herokuAppPage.fileUploadLink.click();
        WaitHelper.shortWait();
    }

    @When("user uploads a file called {string}")
    public void user_uploads_a_file_called(String fileName) {
        this.uploadFileName = fileName;

        String basePath = System.getProperty(Constants.USER_DIRECTORY);
        String filePath = Paths.get(basePath, "src", "test", "resources", "testData",
fileName).toString();
        herokuAppPage.choseFileButton.sendKeys(filePath);
        herokuAppPage.uploadButton.click();
        WaitHelper.shortWait();
```

```
    }

    @Then("the file is successfully uploaded")
    public void the_file_is_successfully_uploaded() {
        String expectedFileName= uploadFileName;
        String actualFileName = herokuAppPage.uploadedFilesResult.getText();
        Assert.assertEquals("File name verification failed", expectedFileName,
actualFileName);
        WaitHelper.shortWait();

    }
//=========DOWNLOAD======
    @Given("clicks on file download page")
    public void clicks_on_file_download_page() {
        herokuAppPage.fileDownloadLink.click();
        WaitHelper.shortWait();
    }

    @When("user clicks on {string}")
    public void user_clicks_on(String fileName) {
        this.downloadFileName = fileName;
        herokuAppPage.LambdaTestTextFileLink.click();
        //herokuAppPage.SomeTextFileLink.click();
        WaitHelper.shortWait();
    }

    @Then("the file is successfully downloaded")
    public void the_file_is_successfully_downloaded() {
        String basePath = System.getProperty(Constants.USER_DIRECTORY);
        String filePath =  Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", downloadFileName).toString();

        File expectedFile = new File(filePath);
        boolean fileExists =expectedFile.exists() && expectedFile.isFile();
        WaitHelper.shortWait();
        Assert.assertTrue("The file existence verification failed.", fileExists);
    }


   @Then("the file content matches the {string}")
 public void the_file_content_matches_the(String expectedFileName) {
       String basePath = System.getProperty(Constants.USER_DIRECTORY);
       String actualDownloadedFilePath = Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", downloadFileName).toString();
        String expectedDownloadedFilePath = Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", expectedFileName).toString();
       WaitHelper.shortWait();

        File actualDownloadedFile = new File(actualDownloadedFilePath);
       File expectedFile = new File(expectedDownloadedFilePath);
       boolean filesContentEquals;
       try {
            filesContentEquals= FileUtils.contentEquals(expectedFile,actualDownloadedFile);
       } catch (IOException e) {
           throw new RuntimeException(e);
       }
               Assert.assertTrue("File content verification Failed", filesContentEquals);

       //DELETE THE DOWNLOADED FILE. YOU WILL NOT SEE : LambdaTest.txt under
Downloads Directory.
       Path path = Paths.get(actualDownloadedFilePath);
```

```
        try {
            Files.delete(path);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
}


}
```

Second Method using FileHelper Java class for downloading, matching, deleting file:

```
package ui_automation.utilities;

import org.apache.commons.io.FileUtils;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class FileHelper {
    public static boolean fileExists(String filePath) {
        File file = new File(filePath);
        return file.exists() && file.isFile();
    }

    public static boolean compareFilesOnBytes(String expectedFilePath, String
actualFilePath) {


        File actualFile = new File(actualFilePath);
        File expectedFile = new File(expectedFilePath);


        try {
            return FileUtils.contentEquals(expectedFile, actualFile);
        } catch (IOException e) {
            throw new RuntimeException("An issue occurs when reading one of these files:\n
In File 1: " + actualFilePath + "\n File 2: " + expectedFilePath, e);

        }
    }
    public static void deleteFile(String filePath) {
        Path path = Paths.get(filePath);
        WaitHelper.shortWait();

            try {
                Files.delete(path);
                System.out.println("File deleted successfully: " + filePath);
            } catch (IOException e) {
                throw new RuntimeException("There was an issue while deleting the File (It
does not exist and cannot be deleted) at " + filePath, e);

            }
        }
```

```
    }
```

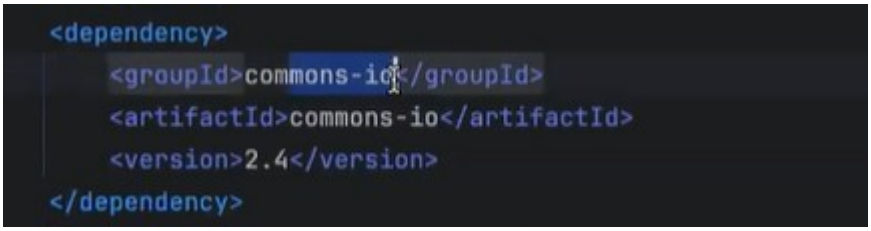
```
package ui_automation.step_definitions;

import constants.Constants;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.apache.commons.io.FileUtils;
import org.junit.Assert;
import org.openqa.selenium.WebDriver;
import ui_automation.pages.HerokuAppPage;
import ui_automation.utilities.ConfigurationReader;
import ui_automation.utilities.Driver;
import ui_automation.utilities.FileHelper;
import ui_automation.utilities.WaitHelper;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class HerokuAppPageSteps {
    private WebDriver driver = Driver.getInstance().getDriver();
    private String downloadFileName;
    private String uploadFileName;
    private HerokuAppPage herokuAppPage = new HerokuAppPage();

    @Given("user navigates to heroku app")
    public void user_navigates_to_heroku_app() {
        String url = ConfigurationReader.getProperty(Constants.UI_CONFIG,
Constants.HEROKU_APP_URL);
        Driver.getInstance().getDriver().get(url);
        WaitHelper.shortWait();

    }

    @Given("clicks on file upload page")
    public void user_clicks_on_file_upload_page() {
        herokuAppPage.fileUploadLink.click();
        WaitHelper.shortWait();
    }

    @When("user uploads a file called {string}")
    public void user_uploads_a_file_called(String fileName) {
        this.uploadFileName = fileName;

        String basePath = System.getProperty(Constants.USER_DIRECTORY);
        String filePath = Paths.get(basePath, "src", "test", "resources", "testData",
fileName).toString();
        herokuAppPage.choseFileButton.sendKeys(filePath);
        herokuAppPage.uploadButton.click();
        WaitHelper.shortWait();
    }
```

```
    @Then("the file is successfully uploaded")
    public void the_file_is_successfully_uploaded() {
        String expectedFileName= uploadFileName;
        String actualFileName = herokuAppPage.uploadedFilesResult.getText();
        Assert.assertEquals("File name verification failed", expectedFileName,
actualFileName);
        WaitHelper.shortWait();

    }
//=========DOWNLOAD======
    @Given("clicks on file download page")
    public void clicks_on_file_download_page() {
        herokuAppPage.fileDownloadLink.click();
        WaitHelper.shortWait();
    }

    @When("user clicks on {string}")
    public void user_clicks_on(String fileName) {
        this.downloadFileName = fileName;
        herokuAppPage.LambdaTestTextFileLink.click();
        //herokuAppPage.SomeTextFileLink.click();
        WaitHelper.shortWait();
    }

    @Then("the file is successfully downloaded")
    public void the_file_is_successfully_downloaded() {
        String basePath = System.getProperty(Constants.USER_DIRECTORY);
        String filePath =  Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", downloadFileName).toString();
        //File expectedFile = new File(filePath);
        //boolean fileExists =expectedFile.exists() && expectedFile.isFile();
        //Second Method using FileHelper file
         boolean fileExists =FileHelper.fileExists(filePath);
         WaitHelper.shortWait();
         Assert.assertTrue("The file existence verification failed.",
fileExists);
    }



   @Then("the file content matches the {string}")
 public void the_file_content_matches_the(String expectedFileName) {
       String basePath = System.getProperty(Constants.USER_DIRECTORY);
       String actualDownloadedFilePath = Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", downloadFileName).toString();
        String expectedDownloadedFilePath = Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", expectedFileName).toString();
       WaitHelper.shortWait();
//First Method
//       File actualDownloadedFile = new File(actualDownloadedFilePath);
//       File expectedFile = new File(expectedDownloadedFilePath);
//       boolean filesContentEquals;
//     try {
//            filesContentEquals=
FileUtils.contentEquals(expectedFile,actualDownloadedFile);
//       } catch (IOException e) {
//           throw new RuntimeException(e);
//       }
       //Assert.assertTrue("File content verification Failed", filesContentEquals);
```

```
// Second Method using FileHelper file:
                boolean filesContentEquals=
FileHelper.compareFilesOnBytes(actualDownloadedFilePath,expectedDownloadedFilePath
);
               Assert.assertTrue("File content verification Failed",
filesContentEquals);
//First Method
       //DELETE THE DOWNLOADED FILE. YOU WILL NOT SEE : LambdaTest.txt under Downloads
Directory.
//       Path path = Paths.get(actualDownloadedFilePath);
//       try {
//           Files.delete(path);
//       } catch (IOException e) {
//           throw new RuntimeException(e);
//       }

       // Second Method using FileHelper file:
        FileHelper.deleteFile(actualDownloadedFilePath);
}


}
```

This library : commons-io helps to deal with files

```
<dependency>
   <groupId>commons-io</groupId>
   <artifactId>commons-io</artifactId>
   <version>2.4</version>
</dependency>
```



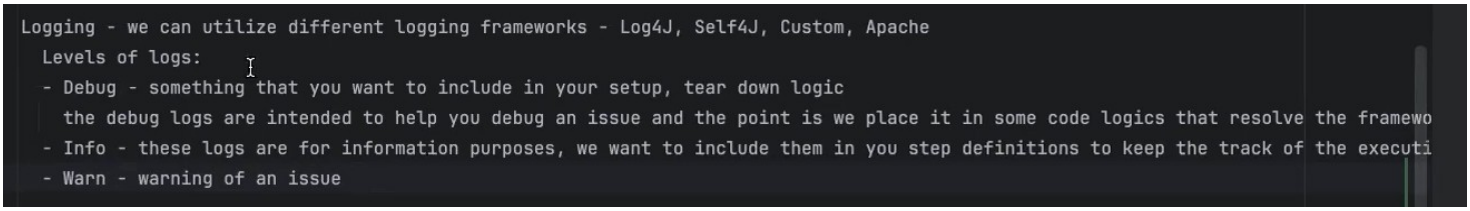
LOG4J

- During the running of test case, user wants some information to be logged in the console.
- We may need the information which helps us to understand the test steps or any failure during the test case execution.
  - E.g: Entered username, entered password, clicked on login button.
  - These above steps get logged. So, you can go and review results after automation is run and finished.
- Log4J provides this detailed level logging, which helps us to review and find out what caused tests to fail. *(Update: In 2021 there was a security vulnerability in Log4J)*
- Log4J has 3 components:
  - **Loggers** → It is responsible for logging information.
  - **Appenders** → it is used to deliver LogEvents to their destination.
    E.gL standard output, file, server etc.
  - **Layouts** → responsible for formatting logging information in different styles.

LOG4J

To setup Log4J:

1. Add Apache Log4J dependency in the POM.XML
   a. `<dependency>`
   b. `<groupId>log4j</groupId>`
   c. `<artifactId>log4j</artifactId>`
   d. `<version>1.2.17</version>`
   e. `</dependency>`
2. Create a new log4j.properties file and save it in the Project root folder.
3. Create an object of Logger class
   a. `static final Logger oLog = LogManager.getLogger(SelectHelper.class);`
4. Use that object to call logger methods such as:
   a. oLog.**info()**
   b. oLog.**warn**()
   c. oLog.**debug**()


Logging - we can utilize different logging frameworks - Log4J, Self4J, Custom, Apache
Levels of logs:
- Debug - something that you want to include in your setup, tear down logic
the debug logs are intended to help you debug an issue and the point is we place it in some code logics that resolve the framewo
- Info - these logs are for information purposes, we want to include them in you step definitions to keep the track of the executi
- Warn - warning of an issue

```java
package ui_automation.step_definitions;

import constants.Constants;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.apache.commons.io.FileUtils;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.junit.Assert;
import org.openqa.selenium.WebDriver;
import ui_automation.pages.HerokuAppPage;
import ui_automation.utilities.ConfigurationReader;
import ui_automation.utilities.Driver;
import ui_automation.utilities.FileHelper;
import ui_automation.utilities.WaitHelper;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class HerokuAppPageSteps {
    private WebDriver driver = Driver.getInstance().getDriver();
    private String downloadFileName;
    private String uploadFileName;
    private HerokuAppPage herokuAppPage = new HerokuAppPage();
    //Log4J Must be Apache
    private final static Logger logger = LogManager.getLogger(HerokuAppPage.class);

    @Given("user navigates to heroku app")
    public void user_navigates_to_heroku_app() {
        String url = ConfigurationReader.getProperty(Constants.UI_CONFIG, Constants.HEROKU_APP_URL);
        Driver.getInstance().getDriver().get(url);
        logger.info("User navigates to "+url);
        WaitHelper.shortWait();

    }

    @Given("clicks on file upload page")
    public void user_clicks_on_file_upload_page() {
        herokuAppPage.fileUploadLink.click();
        logger.info("User succesfully clicks on file upload page");

        WaitHelper.shortWait();
```

```
    }

    @When("user uploads a file called {string}")
    public void user_uploads_a_file_called(String fileName) {
        this.uploadFileName = fileName;

        String basePath = System.getProperty(Constants.USER_DIRECTORY);
        String filePath = Paths.get(basePath, "src", "test", "resources", "testData",
fileName).toString();
        herokuAppPage.choseFileButton.sendKeys(filePath);
        herokuAppPage.uploadButton.click();
           logger.info("User clicked on upload button");
        WaitHelper.shortWait();
    }

    @Then("the file is successfully uploaded")
    public void the_file_is_successfully_uploaded() {
        String expectedFileName= uploadFileName;
        String actualFileName = herokuAppPage.uploadedFilesResult.getText();
        Assert.assertEquals("File name verification failed", expectedFileName,
actualFileName);
           logger.info("File name verification passed !");

        WaitHelper.shortWait();

    }
//=========DOWNLOAD======
    @Given("clicks on file download page")
    public void clicks_on_file_download_page() {
        herokuAppPage.fileDownloadLink.click();
           logger.info("User clicks on file download page");

        WaitHelper.shortWait();
    }

    @When("user clicks on {string}")
    public void user_clicks_on(String fileName) {
        this.downloadFileName = fileName;
        herokuAppPage.LambdaTestTextFileLink.click();
        //herokuAppPage.SomeTextFileLink.click();
           logger.info("User clicks on"+fileName+" file");
        WaitHelper.shortWait();
    }

    @Then("the file is successfully downloaded")
    public void the_file_is_successfully_downloaded() {
        String basePath = System.getProperty(Constants.USER_DIRECTORY);
        String filePath =  Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", downloadFileName).toString();
        //File expectedFile = new File(filePath);
        //boolean fileExists =expectedFile.exists() && expectedFile.isFile();
        //Second Method using FileHelper file
        boolean fileExists =FileHelper.fileExists(filePath);
        WaitHelper.shortWait();
        Assert.assertTrue("The file existence verification failed.", fileExists);
           logger.info("The file is present at: "+filePath);
    }
```

```
   @Then("the file content matches the {string}")
 public void the_file_content_matches_the(String expectedFileName) {
       String basePath = System.getProperty(Constants.USER_DIRECTORY);
       String actualDownloadedFilePath = Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", downloadFileName).toString();
        String expectedDownloadedFilePath = Paths.get(basePath, "src", "test", "resources",
"testData","Downloads", expectedFileName).toString();
       WaitHelper.shortWait();
//First Method
//       File actualDownloadedFile = new File(actualDownloadedFilePath);
//       File expectedFile = new File(expectedDownloadedFilePath);
//       boolean filesContentEquals;
//     try {
//             filesContentEquals=
FileUtils.contentEquals(expectedFile,actualDownloadedFile);
//       } catch (IOException e) {
//           throw new RuntimeException(e);
//       }
       //Assert.assertTrue("File content verification Failed", filesContentEquals);

// Second Method using FileHelper file:
               boolean filesContentEquals=
FileHelper.compareFilesOnBytes(actualDownloadedFilePath,expectedDownloadedFilePath);
             Assert.assertTrue("File content verification Failed", filesContentEquals);
        logger.info("The files content are identical");
//First Method
       //DELETE THE DOWNLOADED FILE. YOU WILL NOT SEE : LambdaTest.txt under Downloads
Directory.
//       Path path = Paths.get(actualDownloadedFilePath);
//       try {
//           Files.delete(path);
//       } catch (IOException e) {
//           throw new RuntimeException(e);
//       }

       // Second Method using FileHelper file:
       FileHelper.deleteFile(actualDownloadedFilePath);
        logger.info("The file succesfully deleted");
}


}
```